GBDT

In [2]:

```
# Load the Drive helper and mount
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:

```
%env JOBLIB_TEMP_FOLDER=/tmp
```

In [0]:

```
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

# using the SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, con)

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
```

```
#Changing reviews with Score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print(filtered_data.shape)
```

In [0]:

```
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=False)
final.shape
```

In [0]:

```
final.head(2)
```

In [0]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

In [0]:

```
final.sort_values('Time',axis=0,ascending=True,inplace=True,kind='quicksort')
final.head(2)
```

In [0]:

```
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [0]:

```
#Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTML tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 1:
                        all_positive_words.append(s) #list of all words used to describe positive r
eviews
                    if(final['Score'].values)[i] == 0:
                        all_negative_words.append(s) #list of all words used to describe negative r
eviews reviews
```

```
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("*****************************************************************")

    final_string.append(str1)
    i+=1
```

In [0]:

```
final['CleanedText']=final_string #adding a column of CleanedText which displays the data after pr
e-processing of the review
final['CleanedText']=final['CleanedText'].str.decode("utf-8")
final.head(3)
```

In [0]:

```
import pickle
pickle.dump(final, open('final.p', 'wb'))
#final_sent = pickle.load(open('data.p','rb'))
final.shape
```

In [9]:

```
import pickle
final = pickle.load(open('drive/My Drive/Colab Notebooks/gbdt/final.p','rb'))


from sklearn.model_selection import train_test_split
##Sorting data according to Time in ascending order for Time Based Splitting
time_sorted_data = final.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort
', na_position='last')
final.head(2)
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summ
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	939340800	EV bo educati
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	940809600	This w seri great to sp time

In [5]:

```
y = final['Score'].iloc[:100000]
x = final['CleanedText'].iloc[:100000]
x.shape,y.shape
```

Out[5]:

```
((100000,), (100000,))
```

BOW

In [6]:

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

X_tra, X_tes, y_train, y_test = train_test_split(x.values,y.values,test_size=0.3,random_state=0)

#Implementing BAG of words
bow = CountVectorizer(ngram_range=(0,1))
X_tf =bow.fit_transform(X_tra)

# Standerdising the data
norm = StandardScaler(with_mean = False)
X_train = norm.fit_transform(X_tf)

# tfidf test
X_tfte = bow.transform(X_tes)

# Standerdising the data
X_test = norm.transform(X_tfte)



X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:475: DataConversionWarning: Dat
a with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Out[6]:

```
((70000, 31572), (70000,), (30000, 31572), (30000,))
```

In [0]:

```

```

In [7]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV

tscv = TimeSeriesSplit(n_splits=3)
param_distributions = {'learning_rate':[0.001,0.01,0.1,1], 'n_estimators':[50,100,150,200],'max_dep
th':[1,2,3,4]}
gbdt=GradientBoostingClassifier()
rsv = RandomizedSearchCV(gbdt,param_distributions,cv=tscv,scoring="f1",n_jobs=-1,pre_dispatch=2)
rsv.fit(X_train,y_train)

print("Best HyperParameter: ",rsv.best_params_)
print("Best f1: %.2f%%"%(rsv.best_score_*100))
```

```
Best HyperParameter:  {'n_estimators': 200, 'max_depth': 2, 'learning_rate': 1}
Best f1: 94.82%
```

In [6]:

```
%env JOBLIB_TEMP_FOLDER=/tmp
```

```
env: JOBLIB_TEMP_FOLDER=/tmp
```

In [0]:

```
from sklearn.ensemble import GradientBoostingClassifier
model = GradientBoostingClassifier(n_estimators= 200, max_depth= 2,learning_rate= 1)
model.fit(X_train,y_train)
y_pred = model.predict(X_train)
```

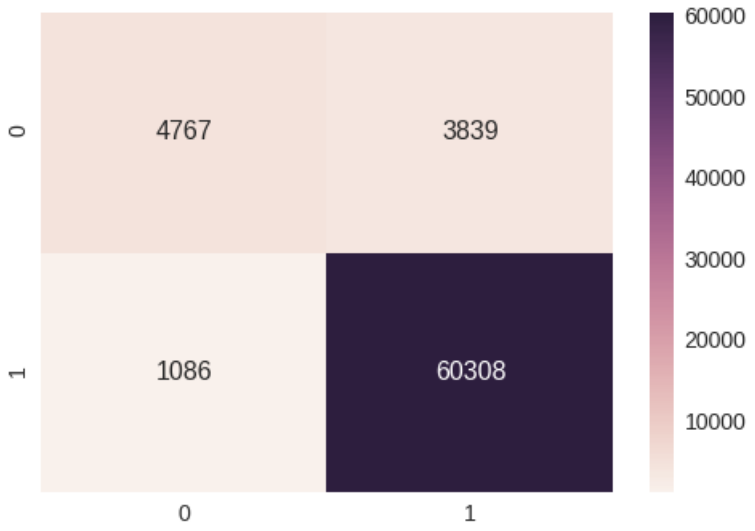
In [9]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns

print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_train, y_pred)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_train, y_pred)*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_train, y_pred)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_train, y_pred)*100))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
result = confusion_matrix(y_train,y_pred)
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 92.964%
F1-Score on the set: 96.077%
Precision_score on test set: 94.015%
Recall_score on test set: 98.231%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 4767  3839]
 [ 1086 60308]]
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b7fbb7fd0>
```



In [0]:

```
X_test.shape
```

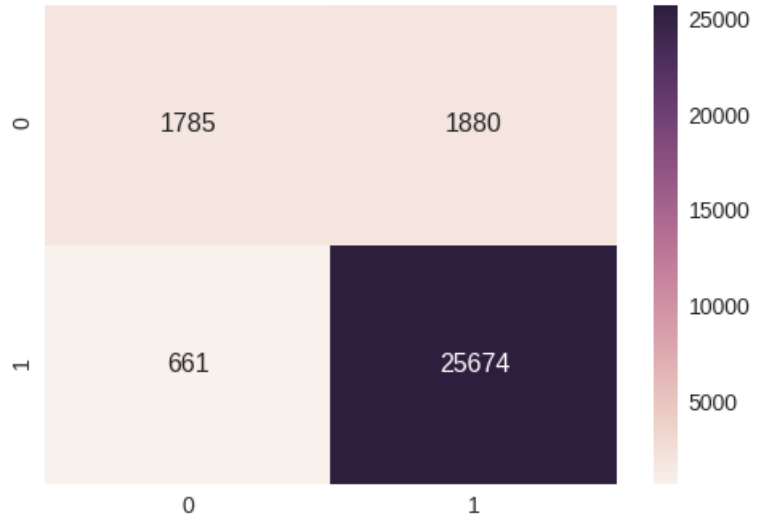
In [10]:

```
y_pred1 = (model.predict(X_test))
print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_test, y_pred1)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_test, y_pred1)*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_test, y_pred1)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_test, y_pred1)*100))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
result = confusion_matrix(y_test,y_pred1)
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 91.530%
F1-Score on the set: 95.285%
Precision_score on test set: 93.177%
Recall_score on test set: 97.490%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 1785  1880]
 [  661 25674]]
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b7fb95518>
```



In [0]:

```
```

TFIDF

In [10]:

```
y = final['Score'].iloc[:100000]
x = final['CleanedText'].iloc[:100000]
x.shape,y.shape
```

Out[10]:

```
((100000,), (100000,))
```

In [0]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

X_tra, X_tes, y_train, y_test = train_test_split(x,y,test_size=0.3,shuffle=False)

#Implementing BAG of words
tfidf = TfidfVectorizer(ngram_range=(0,1),dtype=float)
X_tf =tfidf.fit_transform(X_tra)

# Standerdising the data
norm = StandardScaler(with_mean = False)
X_train = norm.fit_transform(X_tf)

# tfidf test
X_tfte = tfidf.transform(X_tes)
```

```
# Standardising the data
X_test = norm.transform(X_tfte)


from sklearn.model_selection import TimeSeriesSplit
```

In [13]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV

tscv = TimeSeriesSplit(n_splits=3)
param_distributions = {'learning_rate':[0.001,0.01,0.1,1], 'n_estimators':[50,100,150,200],'max_dep
th':[1,2,3,4]}
gbdt=GradientBoostingClassifier()
rsv =
RandomizedSearchCV(gbdt,param_distributions,cv=tscv,scoring="f1_weighted",n_jobs=-1,pre_dispatch=2
)
rsv.fit(X_train,y_train)

print("Best HyperParameter: ",rsv.best_params_)
print("Best f1: %.2f%%"%(rsv.best_score_*100))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
```

```
Best HyperParameter:  {'n_estimators': 100, 'max_depth': 4, 'learning_rate': 1}
Best f1: 87.97%
```

In [14]:

```
model2 = GradientBoostingClassifier(learning_rate=1,max_depth=4,n_estimators=100)
model2.fit(X_train,y_train)
```

```
y_pred2 = model2.predict(X_train)
model2
```

Out[14]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=1, loss='deviance', max_depth=4,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
```

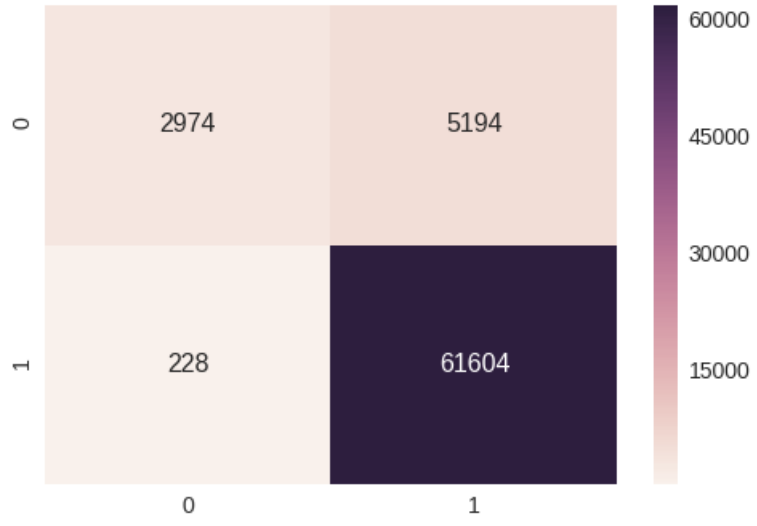
In [15]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns

print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_train, y_pred2)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_train, y_pred2,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_train, y_pred2)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_train, y_pred2)*100))
result = confusion_matrix(y_train,y_pred2)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 92.254%
F1-Score on the set: 95.785%
Precision_score on test set: 92.224%
Recall_score on test set: 99.631%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 2974  5194]
 [  228 61604]]
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b7a710f60>
```



In [16]:

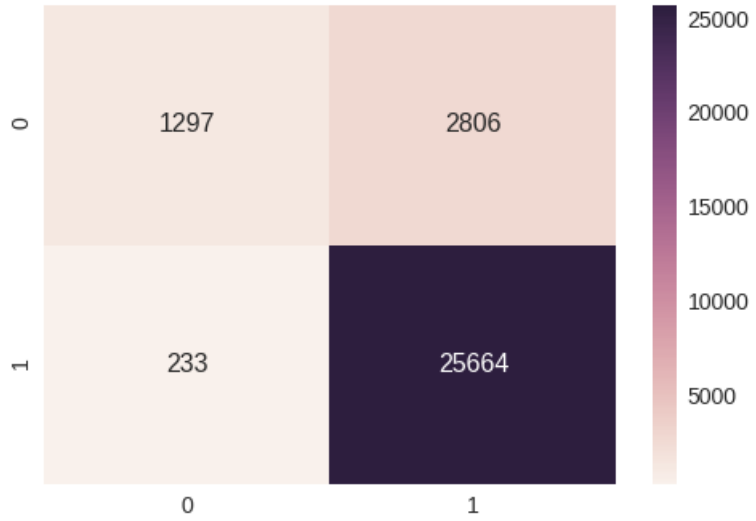
```
y_pred = model2.predict(X_test)
print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_test, y_pred,average='weighted')*100))
```

```
print("Precision_score on test set: %0.3f%%"%(precision_score(y_test, y_pred)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_test, y_pred)*100))
result = confusion_matrix(y_test,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 89.870%
F1-Score on the set: 94.410%
Precision_score on test set: 90.144%
Recall_score on test set: 99.100%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 1297  2806]
 [  233 25664]]
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b7a70fba8>
```



Avg w2c

In [17]:

```
!pip install gensim
```

```
Collecting gensim
  Downloading
https://files.pythonhosted.org/packages/27/a4/d10c0acc8528d838cda5eede0ee9c784caa598dbf40bd0911ff8c
7eb/gensim-3.6.0-cp36-cp36m-manylinux1_x86_64.whl (23.6MB)
    100% |████████████████████████████████| 23.6MB 1.3MB/s
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist-packages (from
gensim) (1.14.6)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.6/dist-packages (from gensim)
(1.11.0)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.6/dist-packages (from
gensim) (0.19.1)
Collecting smart-open>=1.2.1 (from gensim)
  Downloading
https://files.pythonhosted.org/packages/4b/1f/6f27e3682124de63ac97a0a5876da6186de6c19410feab66c1543
055/smart_open-1.7.1.tar.gz
Collecting boto>=2.32 (from smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/23/10/c0b78c27298029e4454a472a1919bde20cb182dab1662cec7f2ca
523/boto-2.49.0-py2.py3-none-any.whl (1.4MB)
    100% |████████████████████████████████| 1.4MB 13.9MB/s
Collecting bz2file (from smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/61/39/122222b5e85cd41c391b68a99ee296584b2a2d1d233e7ee32b453
```

```
f2d/bz2file-0.98.tar.gz
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from smart-
open>=1.2.1->gensim) (2.18.4)
Collecting boto3 (from smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/94/04/c48c102e11b0cb2e3d4a7bdda49647b40e2ae03279ce9ba935e4a
b89/boto3-1.9.34-py2.py3-none-any.whl (128kB)
    100% |████████████████████████████████| 133kB 27.1MB/s
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from
requests->smart-open>=1.2.1->gensim) (2018.10.15)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-packages (from
requests->smart-open>=1.2.1->gensim) (2.6)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /usr/local/lib/python3.6/dist-packages
(from requests->smart-open>=1.2.1->gensim) (1.22)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages
(from requests->smart-open>=1.2.1->gensim) (3.0.4)
Collecting jmespath<1.0.0,>=0.7.1 (from boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/b7/31/05c8d001f7f87f0f07289a5fc0fc3832e9a57f2dbd4d3b0fee70e
365/jmespath-0.9.3-py2.py3-none-any.whl
Collecting s3transfer<0.2.0,>=0.1.10 (from boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/d7/14/2a0004d487464d120c9fb85313a75cd3d71a7506955be458eebfe
b1d/s3transfer-0.1.13-py2.py3-none-any.whl (59kB)
    100% |████████████████████████████████| 61kB 20.1MB/s
Collecting botocore<1.13.0,>=1.12.34 (from boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/91/83/3185727fb3d0204bc2d09ebfbf26bf6725e75f70f35cda477f9b9
61d/botocore-1.12.34-py2.py3-none-any.whl (4.7MB)
    100% |████████████████████████████████| 4.7MB 6.6MB/s
Requirement already satisfied: python-dateutil<3.0.0,>=2.1; python_version >= "2.7" in
/usr/local/lib/python3.6/dist-packages (from botocore<1.13.0,>=1.12.34->boto3->smart-open>=1.2.1->
gensim) (2.5.3)
Collecting docutils>=0.10 (from botocore<1.13.0,>=1.12.34->boto3->smart-open>=1.2.1->gensim)
  Downloading
https://files.pythonhosted.org/packages/36/fa/08e9e6e0e3cbd1d362c3bbee8d01d0aedb2155c4ac112b19ef3ca
d8d/docutils-0.14-py3-none-any.whl (543kB)
    100% |████████████████████████████████| 552kB 23.2MB/s
Building wheels for collected packages: smart-open, bz2file
  Running setup.py bdist_wheel for smart-open ... -   \   done
  Stored in directory:
/root/.cache/pip/wheels/23/00/44/e5b939f7a80c04e32297dbd6d96fa3065af89ecf57e2b5f89f
  Running setup.py bdist_wheel for bz2file ... -   done
  Stored in directory:
/root/.cache/pip/wheels/81/75/d6/e1317bf09bf1af5a30befc2a007869fa6e1f516b8f7c591cb9
Successfully built smart-open bz2file
Installing collected packages: boto, bz2file, jmespath, docutils, botocore, s3transfer, boto3, sma
rt-open, gensim
Successfully installed boto-2.49.0 boto3-1.9.34 botocore-1.12.34 bz2file-0.98 docutils-0.14 gensim
-3.6.0 jmespath-0.9.3 s3transfer-0.1.13 smart-open-1.7.1
```

In [0]:

```
X_tra, X_tes, y_train, y_test = train_test_split(x,y,test_size=0.3,shuffle=False)

sent_of_train=[]
for sent in X_tra:
    sent_of_train.append(sent.split())

sent_of_test=[]
for sent in X_tes:
    sent_of_test.append(sent.split())
```

In [19]:

```
#word to vector
from gensim.models import Word2Vec
w2v_model=Word2Vec(sent_of_train,min_count=3,size=200, workers=4)# words which occurs 3 times; 500
dimensions

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 3 times ",len(w2v_words))
```

```
number of words that occured minimum 3 times  14050
```

In [20]:

```
# compute average word2vec for each review for X_train .
from tqdm import tqdm
import numpy as np



train_vectors = []
for sent in tqdm(sent_of_test):
    sent_vec = np.zeros(200)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)
```

```
100%|██████████| 30000/30000 [00:52<00:00, 569.76it/s]
```

In [21]:

```
train_vectors1 = []
for sent in tqdm(sent_of_train):
    sent_vec = np.zeros(200)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors1.append(sent_vec)
```

```
100%|██████████| 70000/70000 [01:56<00:00, 600.91it/s]
```

In [22]:

```
len(train_vectors),len(train_vectors1)
```

Out[22]:

```
(30000, 70000)
```

In [23]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit

# Data-preprocessing: Standardizing the data
sc = StandardScaler(with_mean = False)
X_train3 = sc.fit_transform(train_vectors1)
X_test3 = sc.transform(train_vectors)

tscv = TimeSeriesSplit(n_splits=4)


y_train.shape,X_train3.shape,X_test3.shape
```

Out[23]:

```
((70000,), (70000, 200), (30000, 200))
```

In [0]:

```
pickle.dump(X_train3, open('X_train3.p', 'wb'))
pickle.dump(X_test3, open('X_test3.p', 'wb'))
```

In [0]:

```
##############################
X_train3 = pickle.load(open('drive/My Drive/Colab Notebooks/gbdt/X_train3.p','rb'))
X_test3 = pickle.load(open('drive/My Drive/Colab Notebooks/gbdt/X_test3.p','rb'))
```

In [25]:

```
%env JOBLIB_TEMP_FOLDER=/tmp
```

```
env: JOBLIB_TEMP_FOLDER=/tmp
```

In [61]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV

tscv = TimeSeriesSplit(n_splits=3)
param_distributions = {'learning_rate':[0.001,0.01,0.1,1], 'n_estimators':[50,100,150,200],'max_dep
th':[1,2,3,4]}
gbdt=GradientBoostingClassifier()
rsv =
RandomizedSearchCV(gbdt,param_distributions,cv=tscv,scoring="f1_weighted",n_jobs=-1,pre_dispatch=2
)
rsv.fit(X_train3,y_train)

print("Best HyperParameter: ",rsv.best_params_)
print("Best f1: %.2f%%"%(rsv.best_score_*100))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
```

```
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
```

```
Best HyperParameter:  {'n_estimators': 150, 'max_depth': 1, 'learning_rate': 1}
Best f1: 82.04%
```

In [64]:

```
model3 = GradientBoostingClassifier(learning_rate=1,max_depth=1,n_estimators=150)
model3.fit(X_train3,y_train)
y_pred3 = model3.predict(X_train3)
model3
```

Out[64]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=1, loss='deviance', max_depth=1,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=150,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
```

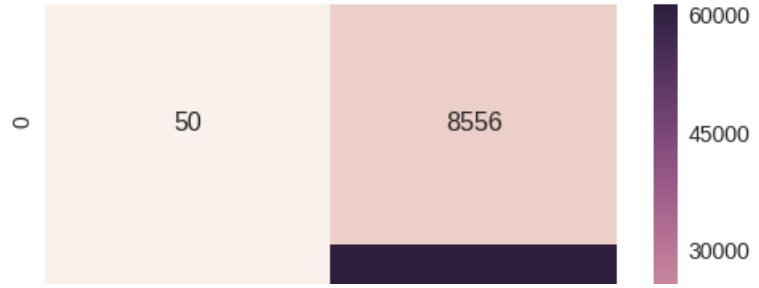
In [65]:

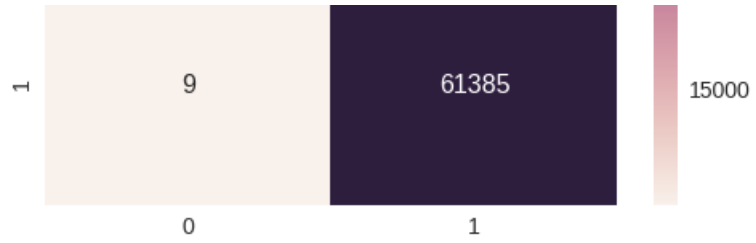
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns

print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_train, y_pred3)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_train, y_pred3,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_train, y_pred3)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_train, y_pred3)*100))
result = confusion_matrix(y_train,y_pred3)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 87.764%
F1-Score on the set: 82.128%
Precision_score on test set: 87.767%
Recall_score on test set: 99.985%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[   50  8556]
 [    9 61385]]
```

Out[65]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b68b6b470>
```





In [66]:

```
y_pred = model3.predict(X_test3)
model3
```

Out[66]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=1, loss='deviance', max_depth=1,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=150,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
```

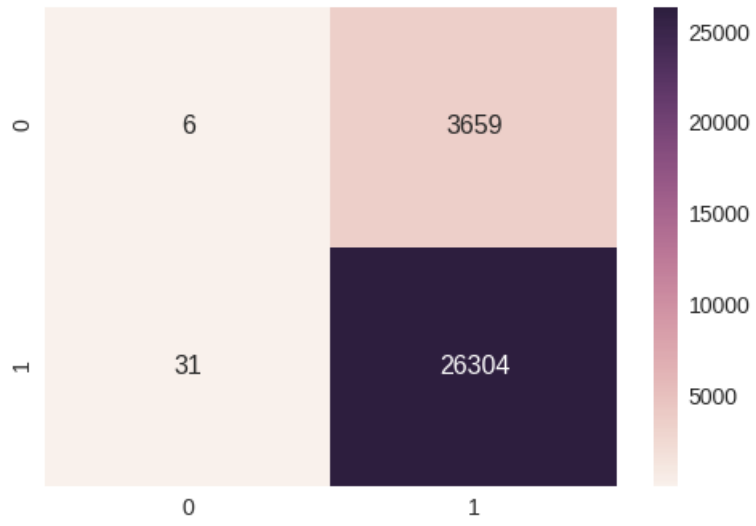
In [67]:

```
print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_test, y_pred,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_test, y_pred)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_test, y_pred)*100))
result = confusion_matrix(y_test,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 87.700%
F1-Score on the set: 82.069%
Precision_score on test set: 87.788%
Recall_score on test set: 99.882%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[    6  3659]
 [   31 26304]]
```

Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b68b66b70>
```



AVG TF-IDF W2V

In [37]:

```
y = final['Score'].iloc[:100000]
x = final['CleanedText'].iloc[:100000]
x.shape,y.shape
```

Out[37]:

```
((100000,), (100000,))
```

In [0]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

X_tra, X_tes, y_train, y_test = train_test_split(x.values,y.values,test_size=0.3,random_state=0)
```

In [0]:

```
sent_of_train=[]
for sent in X_tra:
    sent_of_train.append(sent.split())

sent_of_test=[]
for sent in X_tes:
    sent_of_test.append(sent.split())
```

In [40]:

```
#word to vector
from gensim.models import Word2Vec
w2v_model=Word2Vec(sent_of_train,min_count=3,size=200, workers=4)# words which occurs 3 times; 500 dimensions

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 3 times ",len(w2v_words))
```

```
number of words that occured minimum 3 times  14075
```

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
m = TfidfVectorizer()
tf_idf_matrix = m.fit_transform(X_tra)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(m.get_feature_names(), list(m.idf_)))
```

In [42]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = m.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(sent_of_train): # for each review/sentence
    sent_vec = np.zeros(200) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
```

```
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 70000/70000 [02:17<00:00, 510.86it/s]
```

In [43]:

```
tfidf_sent_vectors1 = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(sent_of_test): # for each review/sentence
    sent_vec = np.zeros(200) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors1.append(sent_vec)
    row += 1
```

```
100%|██████████| 30000/30000 [01:01<00:00, 491.37it/s]
```

In [0]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit

# Data-preprocessing: Standardizing the data
sc = StandardScaler(with_mean = False)
X_train4 = sc.fit_transform(tfidf_sent_vectors)
X_test4 = sc.transform(tfidf_sent_vectors1)
```

In [45]:

```
X_train4.shape,X_test4.shape
```

Out[45]:

```
((70000, 200), (30000, 200))
```

In [0]:

```
pickle.dump(X_train4, open('X_train4.p', 'wb'))
pickle.dump(X_test4, open('X_test4.p', 'wb'))
```

In [52]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV

tscv = TimeSeriesSplit(n_splits=3)
param_distributions = {'learning_rate':[0.001,0.01,0.1,1], 'n_estimators':[50,100,150,200],'max_depth':[1,2,3,4]}
gbdt=GradientBoostingClassifier()
rsv =
RandomizedSearchCV(gbdt,param_distributions,cv=tscv,scoring="f1_weighted",n_jobs=-1,pre_dispatch=2)
rsv.fit(X_train4,y_train)

print("Best HyperParameter: ",rsv.best_params_)
print("Best f1: %.2f%%"%(rsv.best_score_*100))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted sa
mples.
  'precision', 'predicted', average, warn_for)
```

```
Best HyperParameter:  {'n_estimators': 150, 'max_depth': 1, 'learning_rate': 1}
Best f1: 88.08%
```

In [58]:

```
model4 = GradientBoostingClassifier(learning_rate= 1,max_depth=1,n_estimators=150)
model4.fit(X_train4,y_train)
y_pred = model4.predict(X_train4)
model
```

Out[58]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=1, loss='deviance', max_depth=2,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=200,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
```

In [59]:

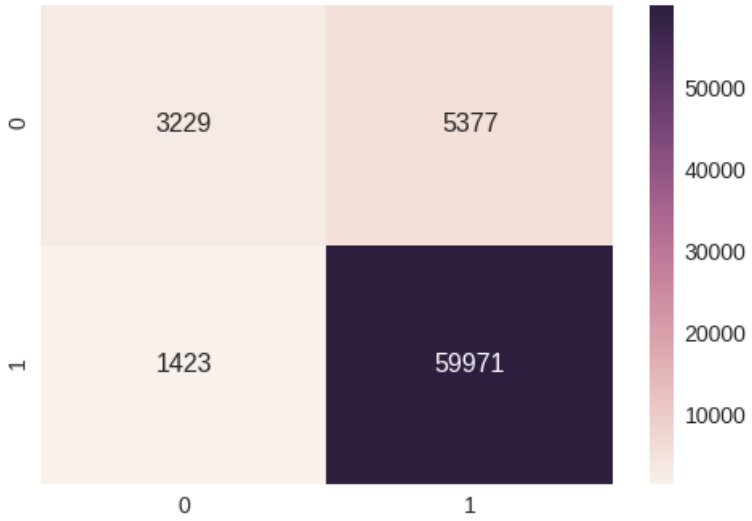
```
print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_train, y_pred)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_train, y_pred,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_train, y_pred)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_train, y_pred)*100))
result = confusion_matrix(y_train,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 90.286%
F1-Score on the set: 88.989%
Precision_score on test set: 91.772%
```

```
Recall_score on test set: 97.682%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 3229  5377]
 [ 1423 59971]]
```

Out[59]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b68cd85c0>
```



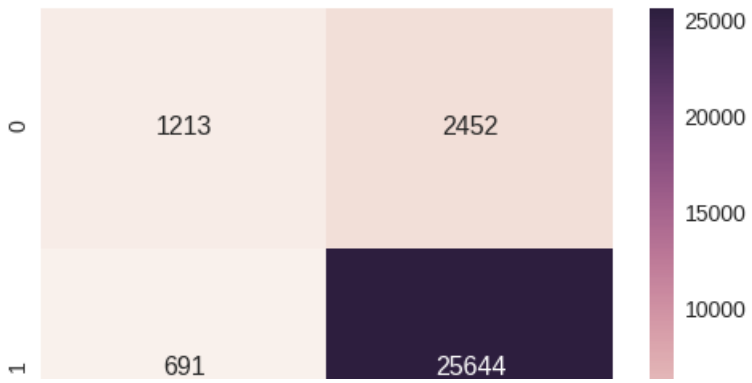
In [60]:

```
y_pred = model4.predict(X_test4)
print("Accuracy on the set: %0.3f%%"%(accuracy_score(y_test, y_pred)*100))
print("F1-Score on the set: %0.3f%%"%(f1_score(y_test, y_pred,average='weighted')*100))
print("Precision_score on test set: %0.3f%%"%(precision_score(y_test, y_pred)*100))
print("Recall_score on test set: %0.3f%%"%(recall_score(y_test, y_pred)*100))
result = confusion_matrix(y_test,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on the set: 89.523%
F1-Score on the set: 88.036%
Precision_score on test set: 91.273%
Recall_score on test set: 97.376%
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 1213  2452]
 [  691 25644]]
```

Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9b66ce2ba8>
```



From the above model, Prefer to choose the best model with stable F1 score

In [7]:

```
import pandas as pd
df = pd.read_csv('drive/My Drive/Colab Notebooks/gbdt/ei09q-9q3dg.csv')
df
```

Out[7]:

	Unnamed: 0	F1 score of Train	F1 score of Test
0	BOW	96.07%	95.28%
1	TFIDF	95.78%	94.41%
2	AVG-W2V	82.12%	82.06%
3	TFIDF-W2V	88.98%	88.03%

TFIDF-W2V has best F1 score on train and test